StructBoost: Boosting Methods for Predicting Structured Output Variables

Chunhua Shen, Guosheng Lin, Anton van den Hengel

Abstract—Structured learning has found many applications in computer vision recently. Analogues to structured support vector machines (SSVM), here we propose boosting algorithms for predicting multivariate or structured outputs, which is referred to as StructBoost. As SSVM generalizes SVM, our StructBoost generalizes standard boosting such as AdaBoost, or LPBoost to structured learning. AdaBoost, LPBoost and many other conventional boosting methods arise as special cases of StructBoost. The resulting optimization problem of StructBoost is more challenging than SSVM in the sense that the problem of StructBoost can involve exponentially many variables and constraints. In contrast, for SSVM one usually has an exponential number of constraints and a cutting-plane method is used. In order to efficiently solve StructBoost, we propose an equivalent 1-slack formulation and solve it using a combination of cutting planes and column generation.

We show the versatility and usefulness of StructBoost on a few problems such as hierarchical multi-class classification, robust visual tracking and image segmentation. In particular, we train a tracking-by-detection based object tracker using the proposed structured boosting. Tracking is implemented as structured output prediction by maximizing the Pascal image area overlap criterion. We show that the structural tracker not only significantly outperforms conventional classification based trackers that do not directly optimize the Pascal image overlap criterion, but also outperforms many other state-of-the-art trackers on the tested videos.

Index Terms—Boosting, AdaBoost, structured learning, conditional random field, image segmentation, object tracking.

CONTENTS

The authors are with School of Computer Science, and Australian Center for Visual Technologies, The University of Adelaide, Adelaide, SA 5005, Australia. Email: chunhua.shen@adelaide.edu.au.

1 INTRODUCTION

Structured learning has attracted extensive attention recently in machine learning and computer vision [1]–[4]. Conventional supervised learning such as classification and regression is the problem of learning a function that predicts the best value for a response variable $y \in \mathbb{R}$ for an input $\boldsymbol{x}$ by making use of a sample of input-output pairs. In many applications, however, the outputs are often complex and cannot be well represented by a scalar because the classes may have inter-class dependencies, or the classes are objects (vectors, sequences, trees, etc.). These problems are referred to as *structured output prediction*. Structured support vector machines (SSVM) [4] generalize the multi-class SVM of [5] and [6] to the much broader problem of learning for interdependent and structured outputs. SSVM uses discriminant functions that take advantage of the dependencies and structure of outputs. In SSVM, the general form of the learned discriminant function is $F(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{w}) : \mathcal{X} \times \mathcal{Y} \mapsto \mathbb{R}$ over input-output pairs and the prediction is achieved by maximizing $F(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{w})$ over all possible $\boldsymbol{y} \in \mathcal{Y}$. As in standard SVM, here $F(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{w})$ is usually defined by a feature mapping function that is only available in the format of inner production, unless the feature mapping function is linear.

Boosting algorithms linearly combine a set of moderately accurate weak learners to form a highly accurate strong predictor. Recently, Shen and Hao proposed a direct formulation for multi-class boosting using the loss functions of multi-class SVM [5], [6]. Inspired by the general boosting framework of [7], they implemented multi-class boosting with the column generation technique. Here we go further by generalizing multi-class boosting of Shen and Hao to broad structured out prediction problems. The advantage of the proposed StructBoost over SSVM might be that in some cases, one wants to learn *sparse and explicit features* for a particular problem. The feature space induced by a nonlinear kernel in SVM—either the standard SVM or SSVM—is usually of large (or even infinite) dimensionality. When the data can be separated with a few features, a kernel-induced feature scheme may still have to use all the features due to the lack of feature selection capability. In contrast, boosting with appropriate weak learners, e.g., decision stumps or decision tress, can select relevant features. In this case, the learning procedure of boosting is also a procedure of feature induction. Moreover, it is in general difficult to derive explicit expressions for kernel-induced features, while boosting's feature induction procedure explicitly introduces nonlinear features into the learned model. The model learned by boosting is usually simpler and computationally more efficient. This is very important for real-time vision applications like object detection and tracking.

1.1 Main contributions

Overall, the main contributions of this work are four-fold.

- To our knowledge, our StructBoost is the first practical boosting method for predicting a broad range of structured outputs. We discuss special cases of this general structured learning framework, including multi-class classification, ordinal regression, optimization of complex measure such as the Pascal image overlap criterion and conditional random field (CRF) parameters learning for image segmentation.
- To implement StructBoost, we adapt the efficient cutting-plane method—originally designed for efficient linear SVM training [8]—for our purpose. We equivalently reformulate the m-slack optimization to 1-slack optimization. We demonstrate that even conventional LPBoost [9] can benefit from this reformulation to gain significant speedup in training.
- We also introduce a new formulation of multi-class boosting, which can be easily implemented by StructBoost. Experiments show encouraging accuracy with faster training time. Also for the first time, we train multi-class boosting classifiers by considering the hierarchical category structure and optimizing the tree loss. This has potential application in object recognition on datasets like ImageNet[1].
- We apply the proposed StructBoost to some computer vision applications and show that our StructBoost can indeed advance some important computer vision problems. In particular, we demonstrate a state-of-the-art object tracker trained by our StructBoost. We also demonstrate an application for CRF and superpixel based image segmentation. We use StructBoost together with graph cuts for CRF parameter learning.

Since our StructBoost builds upon the fully corrective boosting of Shen and Li [7], it inherits the desirable properties of column generation based boosting, such as a fast convergence rate and a clear explanation from the primal-dual convex optimization perspective.

1.2 Related work

The two state-of-the-art structured learning methods are CRF [10] and SSVM [4], which captures the interdependency among output variables. The significance of CRF is in the global training for structured prediction as a convex optimization problem. SSVM follows this path but employs a different loss function (hinge loss) and optimization methods. Our StructBoost is directly inspired by SSVM. StructBoost is an extension of boosting methods to structured prediction. It therefore builds upon the work of column generation boosting [7] and the direct formulation for multi-class boosting [11]. Indeed, we show the multi-class boosting of [11] is a special case of the general framework presented here.

CRF and SSVM have been applied to various problems in machine learning and computer vision mainly because the learned models can easily integrate prior knowledge given a problem of interest. For example, the linear chain CRF widely used in natural language processing estimates sequences of labels for sequences of input samples due to the fact that CRF can take context into account [10], [12].

1. http://www.image-net.org/

SSVM achieves so based on the joint feature maps over the input-output pairs, where features can be represented equivalently as in CRF [8]. CRF is particularly of interest in computer vision for its success in semantic image segmentation [13]. A critical issue of semantic image segmentation is to integrate local and global features for the prediction of local pixel/segment labels. Semantical segmentation is achieved by exploiting the class information with a CRF model. SSVM can also be used for similar purposes as demonstrated in [14]. Blaschko and Lampert [3] trained SSVM models to predict the bounding box of objects in a given image, by optimizing the Pascal bounding box overlap score. The work in [1] introduced structured learning to real-time object detection and tracking, which also optimizes the Pascal box overlap score. SSVM has also been used to learn statistics that capture the spatial arrangements of various object classes in images [15]. The trained model can then simultaneously predict a structured labeling of the entire image. Based on the idea of large-margin learning in SSVM, Szummer et al. [16] learned optimal parameters of a CRF, avoiding tedious cross validation. The survey of [2] has provided a comprehensive review of structured learning and its application in computer vision. Next we review some boosting attempts to structured prediction.

There are a few structured boosting methods in the literature. As we discuss here, none of them is as general and practical as ours. Ratliff et al. [17] proposed boosting for imitation learning based on structured prediction called maximum margin planning (MMP). In the MMPBoost of [17], a demonstrated policy is provided as example behavior for training and the purpose is to learn a function over features of the environment that produce policies with similar behavior. Although MMPBoost is structured learning in that the output is a vector, it differs ours fundamentally. First, MMPBoost is *heuristic* because the optimization procedure is not directly defined on the joint function $F(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{w})$. Second, MMPBoost is based on the idea of gradient descent boosting [18], and our StructBoost is built upon fully corrective boosting of Shen and Li [7]. Most importantly, MMPBoost is specifically designed for the planning problem in robotics. It remains unclear how MMPBoost can be extended to other general structured learning problems.

Parker [19] developed a margin-based structured perceptron update and showed that it can incorporate general notions of misclassification cost as well as kernels. Although it is called structured boosting, Parker assumed that the dictionary (weak learners) are known *a priori*, and the only variable to optimize is the coefficient $\boldsymbol{w}$. No weak learner training is involved. Therefore the method in [19] is essentially an online version of SSVM. Wang et al. [20] learned a local predictor using standard methods, e.g., SVM, but then achieved improved structured classification by exploiting the influence of misclassified components after structured prediction, and iteratively re-training the local predictor. Again, this approach is heuristic and it is more like a post-processing procedure—it does not directly optimize the structured learning objective.

1.3 Notation

A bold lowercase letter ($\boldsymbol{u}$, $\boldsymbol{v}$) denotes a column vector. An element-wise inequality between two vectors or matrices like $\boldsymbol{u} \geq \boldsymbol{v}$ means $u_i \geq v_i$ for all i. Let $(\boldsymbol{x}_i; y_i) \in \mathcal{X} \times \mathcal{Y}$, with $\mathcal{X} \subset \mathbb{R}^d$. Unlike classification ($\mathcal{Y} = \{1, 2, \ldots, k\}$) or regression ($\mathcal{Y} = \mathbb{R}$) problems where y_i is either a discrete or real-valued scalar. We are interested in the case where elements of $\mathcal{Y}$ are *structured variables*, e.g., vectors, strings, graphs. We denote $\mathcal{F}$ a set of weak learners (dictionary); the size of $\mathcal{F}$ can be infinite. Each $\hbar_j(\cdot, \cdot) \in \mathcal{F}, j = 1 \ldots n$, is a function that maps an input-output $(\boldsymbol{x}, \boldsymbol{y})$ pair to $\{-1, +1\}$. Although our discussion works for the general case that $\hbar(\cdot, \cdot)$ can be any real value, we consider binary weak learners here. Clearly $\hbar(\cdot, \cdot)$ plays the same role as the feature representation of inputs and outputs $\Phi(\boldsymbol{x}, \boldsymbol{y})$ in SSVM. We define column vectors $\mathbb{h}(\boldsymbol{x}, \boldsymbol{y}) = [\hbar_1(\boldsymbol{x}, \boldsymbol{y}), \hbar_2(\boldsymbol{x}, \boldsymbol{y}), \cdots, \hbar_n(\boldsymbol{x}, \boldsymbol{y})]^\top$ to be the outputs of all weak learners on the training datum $\boldsymbol{x}$ and label $\boldsymbol{y}$. The discriminant function that we want to learn is then $F : \mathcal{X} \times \mathcal{Y} \mapsto \mathbb{R}$ over input-output pairs, which has the form of

$$F(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{w}) = \boldsymbol{w}^\top \mathbb{h}(\boldsymbol{x}, \boldsymbol{y}) = \textstyle\sum_j w_j \hbar_j(\boldsymbol{x}, \boldsymbol{y}), \tag{1}$$

with $\boldsymbol{w} > 0$. Analogue to SSVM, the inference step is to maximize the joint compatibility function over the output $\boldsymbol{y}$:

$$\boldsymbol{y}^\star = \operatorname*{argmax}_{\boldsymbol{y}} F(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{w}) = \operatorname*{argmax}_{\boldsymbol{y}} \boldsymbol{w}^\top \mathbb{h}(\boldsymbol{x}, \boldsymbol{y}). \tag{2}$$

We denote by $\mathbb{1}$ a column vector of all 1's, whose dimension should be clear from the context. $\|\boldsymbol{x}\|_1$ and $\|\boldsymbol{x}\|_2$ denote the ℓ_1 and ℓ_2 norms in the vector space, respectively. Next, we explain how StructBoost works in Section 2, including how to efficiently solve the resulting optimization problem. We then highlight a few applications in various domains in Section 3. Experimental results are shown in Section 4 and we conclude the paper in the last section.

2 Structured Boosting

Before we present the proposed general structured boosting framework, we introduce the general loss for structured learning and then we take a look at some special instances: classification, ordinal regression, optimizing special criteria such as area under the ROC curve and the Pascal image area overlap ratio, and learning CRF parameters using StructBoost.

To measure the accuracy of a prediction, as in SSVM, we want to learn with arbitrary loss functions $\Delta : \mathcal{Y} \times \mathcal{Y} \mapsto \mathbb{R}$. $\Delta(\boldsymbol{y}, \boldsymbol{y}')$ calculates the loss associated with a prediction $\boldsymbol{y}'$ against the true label value $\boldsymbol{y}$. Note that in general we assume $\Delta(\boldsymbol{y}, \boldsymbol{y}) = 0$ and $\Delta(\boldsymbol{y}, \boldsymbol{y}') > 0$ for any $\boldsymbol{y}' \neq \boldsymbol{y}$. We also assume that the loss is upper bounded.

The formulation of StructBoost can be written as (m-slack

primal) with the model defined in (1):

$$\min_{\boldsymbol{w},\boldsymbol{\xi}} \ \|\boldsymbol{w}\|_1 + \tfrac{C}{m}\mathbb{1}^\top\boldsymbol{\xi} \tag{3a}$$

$$\text{s.t.:} \ \boldsymbol{w}^\top\Big[\mathbb{h}(\boldsymbol{x}_i,\boldsymbol{y}_i) - \mathbb{h}(\boldsymbol{x}_i,\boldsymbol{y})\Big] \geq \Delta(\boldsymbol{y}_i,\boldsymbol{y}) - \xi_i, \quad \forall i = 1,\ldots,m; \text{ and } \forall \boldsymbol{y} \in \mathcal{Y}\backslash\boldsymbol{y}_i, \tag{3b}$$

$$\boldsymbol{w} \geq 0; \boldsymbol{\xi} \geq 0. \tag{3c}$$

Here we have used the ℓ_1 norm as the regularization function to control the complexity of the learned model. To simplify the notation, we introduce $\delta\mathbb{h}_i(\boldsymbol{y}) = \mathbb{h}(\boldsymbol{x}_i,\boldsymbol{y}_i) - \mathbb{h}(\boldsymbol{x}_i,\boldsymbol{y})$; and the constraints can be re-written as: $\boldsymbol{w}^\top\delta\mathbb{h}_i(\boldsymbol{y}) \geq \Delta(\boldsymbol{y}_i,\boldsymbol{y}) - \xi_i$. There are two major obstacles to solve problem (3). First, as in conventional boosting, because the possibility of weak learners $\hbar(\cdot,\cdot)$ can be exponentially large or even infinite, the dimension of $\boldsymbol{w}$ can be exponentially large or infinite. So in general we are not able to directly solve for $\boldsymbol{w}$. Second, same as in SSVM, the number of constraints (3b) can be extremely (or infinitely) large. For example, in the case of multi-label or multi-class classification, the label $\boldsymbol{y}_i$ can be represented as a binary vector (or string) and clearly the possible number of $\boldsymbol{y}$ such that $\boldsymbol{y} \neq \boldsymbol{y}_i$ is exponential in the length of the vector, which is $2^{|\mathcal{Y}|}$. In other words, *problem* (3) *can have an extremely or infinitely large number of variables as well as constraints.* This is much more challenging than solving standard boosting or SSVM from the viewpoint of optimization. In standard boosting, one has a large number of variables and in SSVM, one has a large number of constraints.

For the time being, let us put aside the difficulty of the large number of constraints, and focus on how to iteratively solve for $\boldsymbol{w}$ using column generation as in [7], [9]. The Lagrangian of the m-slack primal problem (3) can be written as:

$$L = \|\boldsymbol{w}\|_1 + \tfrac{C}{m}\mathbb{1}^\top\boldsymbol{\xi} - \sum_{i,\boldsymbol{y}\neq\boldsymbol{y}_i}\lambda_{(i,\boldsymbol{y})}\cdot\Big\{\boldsymbol{w}^\top\Big[\mathbb{h}(\boldsymbol{x}_i,\boldsymbol{y}_i) - \mathbb{h}(\boldsymbol{x}_i,\boldsymbol{y})\Big] - \Delta(\boldsymbol{y}_i,\boldsymbol{y}) + \xi_i\Big\} - \boldsymbol{\nu}^\top\boldsymbol{w} - \boldsymbol{\beta}^\top\boldsymbol{\xi}, \tag{4}$$

where $\boldsymbol{\lambda},\boldsymbol{\nu},\boldsymbol{\beta}$ are Lagrange multipliers: $\boldsymbol{\lambda} \geq 0, \boldsymbol{\nu} \geq 0, \boldsymbol{\beta} \geq 0$. We denote by $\lambda_{(i,\boldsymbol{y})}$ the Lagrange dual multiplier associated with the margin constraints (3b) for label $\boldsymbol{y} \neq \boldsymbol{y}_i$ and training pair $(\boldsymbol{x}_i,\boldsymbol{y}_i)$. At optimum, the first derivative of the Lagrangian w.r.t. the primal variables must vanish,

$$\frac{\partial L}{\partial \xi_i} = 0 \implies \frac{C}{m} - \sum_{\boldsymbol{y}\neq\boldsymbol{y}_i}\lambda_{(i,\boldsymbol{y})} - \beta_i = 0 \implies 0 \leq \sum_{\boldsymbol{y}\neq\boldsymbol{y}_i}\lambda_{(i,\boldsymbol{y})} \leq \tfrac{C}{m};$$

and,

$$\frac{\partial L}{\partial \boldsymbol{w}} = 0 \implies \mathbb{1} - \sum_{i,\boldsymbol{y}\neq\boldsymbol{y}_i}\lambda_{(i,\boldsymbol{y})}\delta\mathbb{h}_i(\boldsymbol{y}) - \nu = \mathbf{0} \implies \sum_{i,\boldsymbol{y}\neq\boldsymbol{y}_i}\lambda_{(i,\boldsymbol{y})}\delta\mathbb{h}_i(\boldsymbol{y}) \leq \mathbb{1}.$$

By putting them back into the Lagrangian (4) and we can obtain the dual problem of the m-slack formulation in (3):

$$\max_{\boldsymbol{\lambda}} \ \sum_{i,\boldsymbol{y}\neq\boldsymbol{y}_i}\lambda_{(i,\boldsymbol{y})}\Delta(\boldsymbol{y}_i,\boldsymbol{y}) \tag{5a}$$

$$\text{s.t.:} \ \textstyle\sum_{i,\boldsymbol{y}\neq\boldsymbol{y}_i}\lambda_{(i,\boldsymbol{y})}\delta\mathbb{h}_i(\boldsymbol{y}) \leq \mathbb{1}, \tag{5b}$$

$$0 \leq \textstyle\sum_{\boldsymbol{y}\neq\boldsymbol{y}_i}\lambda_{(i,\boldsymbol{y})} \leq \frac{C}{m}, \forall i = 1,\ldots,m. \tag{5c}$$

The idea of column generation is to split the original problem into two problems: the master problem and the subproblem. The master problem is the original problem with only a subset of variables being considered. The subproblem's task is to add new variables into the master problem. Usually the objective function of the subproblem is the reduced cost of the new variable with respect to the current dual variables. At each iteration, the master problem is solved and we obtain dual variables. With the dual variables we solve the subproblem to generate a new weak learner which corresponds to a new variable in the primal, and we re-solve the master problem until convergence. With the primal-dual pair of (3) and (5) and following the general framework of column generation based boosting [7], [9], we can obtain our StructBoost as follows:

Iterate the following three steps until converge:

1) Solve the subproblem which finds the best weak learner by finding the most violated constraint in the dual:

$$\hbar^\star(\cdot,\cdot) = \operatorname*{argmax}_{\hbar(\cdot,\cdot)} \sum_{i,\boldsymbol{y}\neq\boldsymbol{y}_i}\lambda_{(i,\boldsymbol{y})}\Big[\hbar(\boldsymbol{x}_i,\boldsymbol{y}_i) - \hbar(\boldsymbol{x}_i,\boldsymbol{y})\Big] \tag{6}$$

2) Add the selected weak learner into the master problem and re-solve for $\boldsymbol{w}$.
3) Update the dual variable $\boldsymbol{\lambda}$ (using KKT conditions, for example).

This approach, however, may not be practical because it is difficult to solve the master problem (the reduced problem of (3)), which still can have extremely many constraints due to the set of $\{\boldsymbol{y} \in \mathcal{Y} | \boldsymbol{y} \neq \boldsymbol{y}_i\}$. We show the poor scalability of this approach in the experiment section, even for special cases of binary classification. The direct formulation for multi-class boosting in [11] can be seen as a specific instance of this approach, which is in general very slow.

2.1 1-slack formulation for fast optimization

Inspired by the cutting-plane method for fast training of linear SVM [8], we can equivalently rewrite the above problem into a "1-slack" form so that an efficient cutting-plane method can be employed to solve the optimization

problem in (3):

$$\min_{\boldsymbol{w},\xi} \ \|\boldsymbol{w}\|_1 + C\xi \tag{7a}$$

$$\text{s.t.:} \ \frac{1}{m}\boldsymbol{w}^\top\Big[\sum_{i=1}^m c_i\cdot\delta\mathbb{h}_i(\boldsymbol{y})\Big] \geq \frac{1}{m}\sum_{i=1}^m c_i\Delta(\boldsymbol{y}_i,\boldsymbol{y}) - \xi,$$

$$\forall \boldsymbol{c}\in\{0,1\}^m; \forall \boldsymbol{y}\in\mathcal{Y}\backslash\boldsymbol{y}_i, i=1,\cdots,m, \tag{7b}$$

$$\boldsymbol{w}\geq 0; \xi\geq 0. \tag{7c}$$

The following theorem shows the equivalence of problems (3) and (7).

Theorem 2.1. *A solution of problem* (7) *is also a solution of problem* (3) *and* vice versa. *The connections are:* $\boldsymbol{w}^\star_{(7)} = \boldsymbol{w}^\star_{(3)}$ *and* $\xi^\star_{(7)} = \frac{1}{m}\mathbb{1}^\top\boldsymbol{\xi}^\star_{(3)}$.

Proof: The proof adapts the proof in [8]. Given a fixed $\boldsymbol{w}$, the only variable $\boldsymbol{\xi}_{(3)}$ in (3) can be solved by

$$\xi_{i,(3)} = \max_{\boldsymbol{y},\boldsymbol{y}\neq\boldsymbol{y}_i}\Big\{0, \Delta(\boldsymbol{y}_i,\boldsymbol{y}) - \boldsymbol{w}^\top\delta\mathbb{h}_i(\boldsymbol{y})\Big\}, \forall i.$$

For (7), the optimal $\xi_{(7)}$ given a $\boldsymbol{w}$ can be computed as:

$$\begin{aligned}
\xi_{(7)} &= \frac{1}{m}\max_{\boldsymbol{c},\boldsymbol{y}\neq\boldsymbol{y}_i}\left\{\sum_{i=1}^m c_i\Delta(\boldsymbol{y}_i,\boldsymbol{y}) - \boldsymbol{w}^\top\Big[\sum_{i=1}^m c_i\delta\mathbb{h}_i(\boldsymbol{y})\Big]\right\}\\
&= \frac{1}{m}\sum_{i=1}^m\left\{\max_{c_i\in\{0,1\},\boldsymbol{y}\neq\boldsymbol{y}_i} c_i\Delta(\boldsymbol{y}_i,\boldsymbol{y}) - c_i\boldsymbol{w}^\top\delta\mathbb{h}_i(\boldsymbol{y})\right\}\\
&= \frac{1}{m}\sum_{i=1}^m\max_{\boldsymbol{y}\neq\boldsymbol{y}_i}\Big\{0, \Delta(\boldsymbol{y}_i,\boldsymbol{y}) - \boldsymbol{w}^\top\delta\mathbb{h}_i(\boldsymbol{y})\Big\}\\
&= \frac{1}{m}\mathbb{1}^\top\boldsymbol{\xi}_{(3)}.
\end{aligned}$$

Note that $\boldsymbol{c}\in\{0,1\}^m$ in the above equalities. Clearly the objective functions of both problems coincide for any fixed $\boldsymbol{w}$ and the optimal $\boldsymbol{\xi}_{(3)}$ and $\xi_{(7)}$. □

As demonstrated in [8], cutting-plane methods can be used to solve the 1-slack primal problem (7) efficiently. This 1-slack formulation has been used to train linear SVM in linear time. When solving for $\boldsymbol{w}$, (7) is similar to ℓ_1-norm regularized SVM—except the extra non-negativeness constraint on $\boldsymbol{w}$ in our case.

In order to utilize column generation for designing boosting methods, we need to derive the Lagrange dual of the above 1-slack optimization problem. The Lagrangian of the 1-slack primal problem in (7) can be written as:

$$\begin{aligned}
L =& \|\boldsymbol{w}\|_1 + C\xi - \sum_{\boldsymbol{c},\boldsymbol{y}\neq\boldsymbol{y}_i}\lambda_{(\boldsymbol{c},\boldsymbol{y})}\cdot\left\{\frac{1}{m}\boldsymbol{w}^\top\Big[\sum_{i=1}^m c_i\cdot\delta\mathbb{h}_i(\boldsymbol{y})\Big] - \right.\\
&\left.\frac{1}{m}\sum_{i=1}^m c_i\Delta(\boldsymbol{y}_i,\boldsymbol{y}) + \xi\right\} - \boldsymbol{\nu}^\top\boldsymbol{w} - \beta\xi,
\end{aligned} \tag{8}$$

where $\boldsymbol{\lambda},\boldsymbol{\nu},\beta$ are Lagrange multipliers: $\boldsymbol{\lambda}\geq 0, \boldsymbol{\nu}\geq 0, \beta\geq 0$. We denote by $\lambda_{(\boldsymbol{c},\boldsymbol{y})}$ the Lagrange multiplier associated with the inequality constraints for $\boldsymbol{c}\in\{0,1\}^m$ and $\boldsymbol{y}\neq\boldsymbol{y}_i, i=1\ldots m$. Again, at optimum, the first derivative of the Lagrangian w.r.t. the primal variables must be zeros,

$$\begin{aligned}
\frac{\partial L}{\partial\xi} = 0 \implies& C - \sum_{\boldsymbol{c},\boldsymbol{y}\neq\boldsymbol{y}_i}\lambda_{(\boldsymbol{c},\boldsymbol{y})} - \beta = 0\\
\implies& 0\leq\sum_{\boldsymbol{c},\boldsymbol{y}\neq\boldsymbol{y}_i}\lambda_{(\boldsymbol{c},\boldsymbol{y})}\leq C;
\end{aligned}$$

and,

$$\begin{aligned}
\frac{\partial L}{\partial\boldsymbol{w}} = 0 \implies& \mathbb{1} - \frac{1}{m}\sum_{\boldsymbol{c},\boldsymbol{y}\neq\boldsymbol{y}_i}\lambda_{(\boldsymbol{c},\boldsymbol{y})}\cdot\Big[\sum_{i=1}^m c_i\cdot\delta\mathbb{h}_i(\boldsymbol{y})\Big] = \boldsymbol{\nu}.\\
\implies& \frac{1}{m}\sum_{\boldsymbol{c},\boldsymbol{y}\neq\boldsymbol{y}_i}\lambda_{(\boldsymbol{c},\boldsymbol{y})}\cdot\Big[\sum_{i=1}^m c_i\cdot\delta\mathbb{h}_i(\boldsymbol{y})\Big]\leq\mathbb{1}.
\end{aligned} \tag{9}$$

The dual problem of (7) can be written as:

$$\max_{\boldsymbol{\lambda}} \ \sum_{\boldsymbol{c},\boldsymbol{y}\neq\boldsymbol{y}_i}\lambda_{(\boldsymbol{c},\boldsymbol{y})}\sum_{i=1}^m c_i\Delta(\boldsymbol{y}_i,\boldsymbol{y}) \tag{10a}$$

$$\text{s.t.:} \ \frac{1}{m}\sum_{\boldsymbol{c},\boldsymbol{y}\neq\boldsymbol{y}_i}\lambda_{(\boldsymbol{c},\boldsymbol{y})}\Big[\sum_{i=1}^m c_i\cdot\delta\mathbb{h}_i(\boldsymbol{y})\Big]\leq\mathbb{1}, \tag{10b}$$

$$0\leq\textstyle\sum_{\boldsymbol{c},\boldsymbol{y}\neq\boldsymbol{y}_i}\lambda_{(\boldsymbol{c},\boldsymbol{y})}\leq C. \tag{10c}$$

Here $\boldsymbol{c}$ enumerates all possible $\boldsymbol{c}\in\{0,1\}^m$. So in practice, we solve the 1-slack formulation (primal (7) and dual (10)). The subproblem to find the most violated constraint in the dual form for generating weak learners is:

$$\begin{aligned}
\hbar^\star(\cdot,\cdot) &= \operatorname*{argmax}_{\hbar(\cdot,\cdot)}\sum_{\boldsymbol{c},\boldsymbol{y}\neq\boldsymbol{y}_i}\lambda_{(\boldsymbol{c},\boldsymbol{y})}\sum_i c_i\Big[\hbar(\boldsymbol{x}_i,\boldsymbol{y}_i) - \hbar(\boldsymbol{x}_i,\boldsymbol{y})\Big]\\
&= \operatorname*{argmax}_{\hbar(\cdot,\cdot)}\sum_{i,\boldsymbol{y}\neq\boldsymbol{y}_i}\underbrace{\sum_{\boldsymbol{c}}\lambda_{(\boldsymbol{c},\boldsymbol{y})}c_i}_{:=\mu_{(i,\boldsymbol{y})}}\Big[\hbar(\boldsymbol{x}_i,\boldsymbol{y}_i) - \hbar(\boldsymbol{x}_i,\boldsymbol{y})\Big].
\end{aligned} \tag{11}$$

We have changed the order of summation to have a similar form as in the m-slack case.

2.2 Cutting-plane optimization for solving the 1-slack primal

Despite the extra nonnegative-ness constraint $\boldsymbol{w}\geq 0$ in our case, it is easy to modify the cutting-plane method in [8] for solving our problem (7). For the analysis of the cutting-plane method for optimizing the 1-slack primal, readers may refer to [8] for details.

Algorithm 2 summarizes how the original optimization problem (3) can be solved using cutting planes.

In the experiment section, we empirically show that solving (7) using cutting planes can be orders of magnitude faster than solving (3).

In theory, improved cutting-plane methods such as [21] can also be adapted for solving our optimization problem at each column generation.

The algorithmic implementation of our StructBoost is summarized in Algorithm 1. Line 4 finds the most violated constraint and add a new weak learner to the master problem. Here $\mu_{(i,\boldsymbol{y})}$ defined in (11) plays the role as the sample weights associated to each training sample in conventional

Algorithm 1 Column generation for StructBoost

1: **Input:** training examples $(\boldsymbol{x}_1; \boldsymbol{y}_1), (\boldsymbol{x}_2; \boldsymbol{y}_2), \cdots$; parameter C; termination threshold ϵ_{cg}, and the maximum iteration number.
2: **Initialize:** for each i, $(i = 1, \ldots, m)$, randomly pick any $\boldsymbol{y}_i^{(0)} \in \mathcal{Y}$, initialize $\mu_{(i,\boldsymbol{y})} = C/m$ for $\boldsymbol{y} = \boldsymbol{y}_i^{(0)}$, and $\mu_{(i,\boldsymbol{y})} = 0$ for all $\boldsymbol{y} \in \mathcal{Y} \backslash \boldsymbol{y}_i^{(0)}$.
3: **Repeat**
4: − Find and add a new weak learner $\hbar^\star(\cdot,\cdot)$ by solving:

$$\hbar^\star(\cdot,\cdot) = \operatorname*{argmax}_{\hbar(\cdot,\cdot)} \sum_{i,\boldsymbol{y}\neq\boldsymbol{y}_i} \mu_{(i,\boldsymbol{y})} \big[\hbar(\boldsymbol{x}_i, \boldsymbol{y}_i) - \hbar(\boldsymbol{x}_i, \boldsymbol{y})\big].$$

5: − Call **Algorithm 2** to obtain $\boldsymbol{w}, \xi$; $\boldsymbol{\lambda}$, and $\mathcal{W}$.
6: − Update $\mu_{(i,\boldsymbol{y})} = \sum_{\boldsymbol{c}} \lambda_{(\boldsymbol{c},\boldsymbol{y})} c_i$.
7: **Until** either $\frac{1}{m}\sum_{\boldsymbol{c},\boldsymbol{y}\neq\boldsymbol{y}_i} \lambda_{(\boldsymbol{c},\boldsymbol{y})} \big[\sum_{i=1}^m c_i \cdot \delta\hbar_i^\star(\boldsymbol{y})\big] \leq 1 - \epsilon_{\text{cg}}$, or the maximum iteration is reached.
8: **Output:** the discriminant function $F(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{w}) = \boldsymbol{w}^\top \mathbb{h}(\boldsymbol{x}, \boldsymbol{y})$.

Algorithm 2 Cutting planes for solving the 1-slack primal

1: **Input:** cutting-plane termination threshold ϵ_{cp}, and inputs from Algorithm 1.
2: **Initialize:** working set $\mathcal{W} \leftarrow \emptyset$; $c_i = 1$, $\boldsymbol{y}_i' \leftarrow$ any element in $\mathcal{Y}$, for $i = 1, \ldots, m$.
3: **Repeat**
4: − $\mathcal{W} \leftarrow \mathcal{W} \cup \{(c_1, \ldots, c_m, \boldsymbol{y}_1', \ldots, \boldsymbol{y}_m')\}$.
5: − Obtain primal and dual solutions $\boldsymbol{w}, \xi$; $\boldsymbol{\lambda}$ by solving

$$\begin{aligned} \min_{\boldsymbol{w}\geq 0, \xi \geq 0} \quad & \|\boldsymbol{w}\|_1 + C\xi \\ \text{s.t.:} \quad & \forall (c_1, \ldots, c_m, \boldsymbol{y}_1', \ldots, \boldsymbol{y}_m') \in \mathcal{W}: \\ & \frac{1}{m}\boldsymbol{w}^\top \left[\sum_{i=1}^m c_i \cdot \delta\mathbb{h}_i(\boldsymbol{y}_i')\right] \geq \frac{1}{m}\sum_{i=1}^m c_i \Delta(\boldsymbol{y}_i, \boldsymbol{y}_i') - \xi. \end{aligned}$$

6: − **For** $i = 1, \ldots, m$
7: $\boldsymbol{y}_i' = \operatorname{argmax}_{\boldsymbol{y}} \Delta(\boldsymbol{y}_i, \boldsymbol{y}) - \boldsymbol{w}^\top \delta\mathbb{h}_i(\boldsymbol{y})$;
8: $c_i = \begin{cases} 1 & \Delta(\boldsymbol{y}_i, \boldsymbol{y}_i') - \boldsymbol{w}^\top \delta\mathbb{h}_i(\boldsymbol{y}_i') > 0 \\ 0 & otherwise \end{cases}$
9: **End for**
10: **Until** $\frac{1}{m}\boldsymbol{w}^\top \left[\sum_{i=1}^m c_i \delta\mathbb{h}_i(\boldsymbol{y}_i')\right] \geq \frac{1}{m}\sum_{i=1}^m c_i \Delta(\boldsymbol{y}_i, \boldsymbol{y}_i') - \xi - \epsilon_{\text{cp}}$.
11: **Output:** $\boldsymbol{w}, \xi$; $\boldsymbol{\lambda}$, $\mathcal{W}$.

boosting such as AdaBoost. Lines 4 and 5 then solve the primal problem and update the variables. We can see that the training loop is almost identical to these conventional boosting methods. The following theorem shows the convergence property of Algorithm 1.

Theorem 2.2. *Algorithm 1 makes progress at each column generation iteration; i.e., the objective value decreases at each iteration. Hence, in the limit, Algorithm 1 globally solves the optimization problem* (3) *(or* (7) *due to Theorem 2.1) to a prescribed accuracy.*

Proof: Let us assume that the current solution is a finite subset of weak learners and their corresponding coefficients are $\boldsymbol{w}$. When we add a weak learner that is not in the current subset and resolve the problem and the corresponding $\hat{w}$ is zero, then the objective value and the solution keep unchanged. In this case, we can draw a conclusion that the current selected weak learner and the solution $\boldsymbol{w}$ are optimal.

Now let us assume that the optimality condition is violated. We want to show that we can find a weak learner $\hat{\hbar}(\cdot,\cdot)$ that is not in the current set of weak learners, such that its corresponding coefficient $\hat{w} > 0$ holds. Assume that $\hat{\hbar}(\cdot,\cdot)$ is found by solving (11), and the convergence condition $\frac{1}{m}\sum_{\boldsymbol{c},\boldsymbol{y}\neq\boldsymbol{y}_i} \lambda_{(\boldsymbol{c},\boldsymbol{y})} \big[\sum_{i=1}^m c_i \cdot \delta\hat{\hbar}_i(\boldsymbol{y})\big] \leq 1$ does not hold. In other words, we have $\frac{1}{m}\sum_{\boldsymbol{c},\boldsymbol{y}\neq\boldsymbol{y}_i} \lambda_{(\boldsymbol{c},\boldsymbol{y})} \big[\sum_{i=1}^m c_i \cdot \delta\hat{\hbar}_i(\boldsymbol{y})\big] > 1$.

Now if this $\hat{\hbar}(\cdot,\cdot)$ is added into the master problem and the primal solution is not changed; i.e., $\hat{w} = 0$, then we know that in (9), $\nu = 1 - \frac{1}{m}\sum_{\boldsymbol{c},\boldsymbol{y}\neq\boldsymbol{y}_i} \lambda_{(\boldsymbol{c},\boldsymbol{y})} \big[\sum_{i=1}^m c_i \cdot \delta\hat{\hbar}_i(\boldsymbol{y})\big] < 0$. This contradicts the fact that the Lagrange multiplier ν must be nonnegative.

Therefore, after this weak learner is added into the master problem, its corresponding coefficient $\hat{w}$ must be a nonzero positive value. It means that one more free variable is added into the master problem and re-solving the it must reduce the objective value. That means, a strict decrease in the objective is assured. Hence Algorithm 1 makes progress at each iteration. Moreover, since the optimization problem is convex in $\boldsymbol{w}$, a local solution is global. □

3 Special Cases of StructBoost

We consider a few special cases of the proposed general structured boosting in this section.

3.1 Binary classification

Clearly the standard binary classification LPBoost can be seen as a special case of multi-class classification and of StructBoost as well. We write the 1-slack formulation of LPBoost and solve the 1-slack primal using cutting-plane. The primal is:

$$\min_{\boldsymbol{w},\xi} \|\boldsymbol{w}\|_1 + C\xi \tag{12a}$$

$$\text{s.t.:} \ \frac{1}{m}\boldsymbol{w}^\top \left[\sum_{i=1}^m c_i y_i \mathbb{h}'(\boldsymbol{x}_i)\right] \geq \frac{1}{m}\sum_{i=1}^m c_i - \xi, \tag{12b}$$

$$\forall \boldsymbol{c} \in \{0,1\}^m, \forall i = 1, \ldots, m; \ \boldsymbol{w} \geq 0; \xi \geq 0. \tag{12c}$$

Here $y_i \in \{-1, 1\}$ and we define the symbol

$$\mathbb{h}'(\boldsymbol{x}) = [\hbar_1(\boldsymbol{x}) \cdots \hbar_n(\boldsymbol{x})]^\top, \tag{13}$$

which is the outputs of all binary weak classifiers on example $\boldsymbol{x}$. The dual problem of (12) can be easily derived. We show in the experiments that at each iteration of LPBoost, solving (12) is much faster than solving the m-slack primal or dual as shown in [9].

3.2 Multi-class boosting

We first show the MultiBoost algorithm in Shen and Hao [11] can be implemented by the StructBoost framework as follows. We then introduce a new multi-class boosting algorithm. Let $\mathcal{Y} = \{1, 2, \ldots, k\}$ and $\boldsymbol{w} = \boldsymbol{w}_1 \odot \cdots \odot \boldsymbol{w}_k$. Here $\odot$ stacks two vectors. As in [11], $\boldsymbol{w}_y$ is the model parameter associated with the y-th class. The multi-class discriminant function in [11] writes $F(\boldsymbol{x}, y; \boldsymbol{w}) = \boldsymbol{w}_y^\top \mathbb{h}'(\boldsymbol{x})$. Now let us define the orthogonal label coding vector:

$$\Gamma(y) = [\mathbb{I}(y,1), \mathbb{I}(y,2), \cdots, \mathbb{I}(y,k)]^\top \in \{0,1\}^k. \tag{14}$$

Here $\mathbb{I}(y,k)$ is the indicator function defined as:

$$\mathbb{I}(y,k) = \begin{cases} 1 & \text{if } y = k, \\ 0 & \text{if } y \neq k. \end{cases} \tag{15}$$

Then $\mathbb{h}(\boldsymbol{x},y) = \mathbb{h}'(\boldsymbol{x}) \otimes \Gamma(y)$ recovers the StructBoost formulation (3) for multi-class boosting. The operator $\otimes$ calculates the tensor product.

Now we propose a new multi-class boosting algorithm. Instead of learning k model parameter (one $\boldsymbol{w}_r$ for each class) as in Shen and Hao [11], we learn a single parameter $\boldsymbol{w}$. Classes are distinguished by augmenting the data with the label. Let us define label-augmented data as $\boldsymbol{x}'_y = \boldsymbol{x} \otimes \Gamma(y)$, with $\boldsymbol{x}$ the original input data. Clearly the label-augmented data $\boldsymbol{x}'_y$ have the same number of non-zero entries as the original data $\boldsymbol{x}$. This is desirable since the label-augmented data do not increase the computation complexity much by using the sparse data structure. So we formulate the multi-class learning as

$$\min_{\boldsymbol{w},\boldsymbol{\xi}} \ \|\boldsymbol{w}\|_1 + \tfrac{C}{m}\mathbb{1}^\top\boldsymbol{\xi} \tag{16a}$$

$$\text{s.t.:} \ \boldsymbol{w}^\top\left[\mathbb{h}'(\boldsymbol{x}'_{i,y_i}) - \mathbb{h}'(\boldsymbol{x}'_{i,y})\right] \geq 1-\xi_i,$$
$$\forall i = 1,\ldots,m; \text{ and } \forall y \in \{1,\ldots,k\}\backslash y_i, \tag{16b}$$

$$\boldsymbol{w} \geq 0; \boldsymbol{\xi} \geq 0; \tag{16c}$$

with $\mathbb{h}'(\cdot)$ defined in (13). So we only need to set $\mathbb{h}(\boldsymbol{x},y) = \mathbb{h}'(\boldsymbol{x}'_y) = \mathbb{h}'(\boldsymbol{x} \otimes \Gamma(y))$ and $\Delta(y,y') = 1$ to implement this new multi-class boosting in the StructBoost framework. The main difference between (16) and MultiBoost in [11] is that here $\boldsymbol{w} \in \mathbb{R}^n$, while $\boldsymbol{w} \in \mathbb{R}^{n\times k}$ for MultiBoost, with n being the number of weak learners. We compare the performance of this new multi-class boosting in the experiment section.

3.3 Hierarchical multi-class classification

The flexibility of StructBoost allows us to train a multi-class classifier that optimizes the complex tree loss. In many applications such as object categorization, classes of objects are organized in taxonomies or hierarchies. For example, The ImageNet dataset has organized all the classes according to the tree structures of WordNet. This problem is a classification example that the output space has interdependent structures. An example tree structure of image categories is shown in Figure 1.

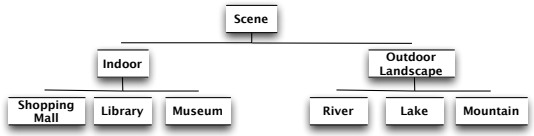
Similar to [4], here we consider the tree loss: $\Delta^{tree}(y,y')$. Given a class tree structure $\mathcal{T}$ with a height of τ (i.e., $\mathcal{T}$ has τ levels), $\Delta^{tree}(y,y')$ is the height of the first common parent node of class y and y' in the tree structure from the bottom to the top. All we need is to redefine the orthogonal coding vector $\Gamma(y)$ in (14), and the algorithmic implementation remains identical as the standard multi-class case that we discussed. We define:

$$\Gamma(y) = \Gamma'(y^{(1)}) \odot \Gamma'(y^{(2)}) \cdots \odot \Gamma'(y^{(\tau)}) \tag{17}$$

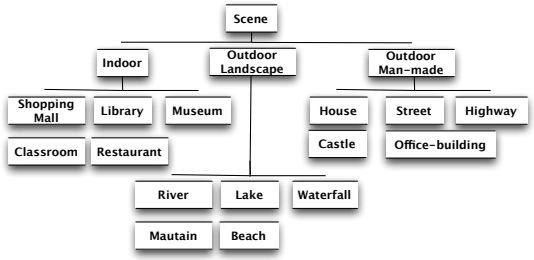
Here $\odot$ stacks two vectors. We define k_l to be the number of classes in the l-th level of the tree structure. $y^{(l)}$ is parent label of y on the l-th level of the tree structure (with $y^{(1)} = y$), and $\Gamma'(y^{(l)}) \in \{0,1\}^{k_l}$ is the orthogonal label coding vector:

$$\Gamma'(y^{(l)}) = [\mathbb{I}(y^{(l)},1), \mathbb{I}(y^{(l)},2), \cdots, \mathbb{I}(y^{(l)},k_l)]^\top. \tag{18}$$

$\mathbb{I}(\cdot,\cdot)$ is defined in (15). The original $\Gamma(y)$ is *flat* in that the inner product $\Gamma(y)^\top\Gamma(y') = 0$ always holds. With the tree loss, $\Gamma(y)^\top\Gamma(y')$ counts the number of common predecessors in the label tree. We have run some experiments on the SUN scene recognition dataset in the experiment section.



(a) Taxonomy of subset 1



(b) Taxonomy of subset 2

Fig. 1: The hierarchical structures of two selected subsets of the SUN dataset [22] used in our experiments for hierarchical image classification.

3.4 Ordinal regression and AUC optimization

In ordinal regression, labels of the training data are ranks. Let us assume that the label $y \in \mathbb{R}$ indicates an ordinal scale, and pairs (i,j) in the set $\mathcal{S}$ has the relationship of example i being ranked higher than j, i.e., $y_i > y_j$. The primal can be written as

$$\min_{\boldsymbol{w},\boldsymbol{\xi}} \ \|\boldsymbol{w}\|_1 + \tfrac{C}{m}\textstyle\sum_{(i,j)\in\mathcal{S}}\xi_{ij} \tag{19a}$$

$$\text{s.t.:} \ \boldsymbol{w}^\top\left[\mathbb{h}'(\boldsymbol{x}_i) - \mathbb{h}'(\boldsymbol{x}_j)\right] \geq 1-\xi_{ij}, \forall (i,j) \in \mathcal{S}, \tag{19b}$$

$$\boldsymbol{w} \geq 0; \boldsymbol{\xi} \geq 0; \tag{19c}$$

Note that (19) also optimizes the area under the receiver operating characteristic (ROC) curve (AUC) criterion. In general, The number of constraints is quadratic in the number of training examples. Directly solving (19) can only solve problems with up to a few thousand training examples. We can reformulate (19) into an equivalent 1-slack problem, same as in (12); and the StructBoost framework can be applied to solve large-scale problems.

3.5 Optimization of the Pascal image overlap criterion

Object detection/localization has used the image area overlap as the loss function [1]–[3], e.g, in the PASCAL object detection challenges:

$$\Delta(\boldsymbol{y},\boldsymbol{y}') = 1 - \frac{\text{area}(\boldsymbol{y} \cap \boldsymbol{y}')}{\text{area}(\boldsymbol{y} \cup \boldsymbol{y}')}, \tag{20}$$

with $\boldsymbol{y}, \boldsymbol{y}'$ being the bounding box coordinates. $\boldsymbol{y} \cap \boldsymbol{y}'$ and $\boldsymbol{y} \cup \boldsymbol{y}'$ are the box intersection and union. In this application, we train the weak learner $\mathbb{h}(\boldsymbol{x}, \boldsymbol{y})$ with the image features extracted from the image patch defined by $\boldsymbol{y}$. For example, we can extract histograms of oriented gradients (HOG) from the image patch $\boldsymbol{y}$ and train a decision stump with the extracted HOG features. This naturally fits into StructBoost.

Note that in this case, to find the most violated constraint in the training step as well as the inference for prediction is in general highly non-convex and it is difficult to find a global solution. In [3], a branch-and-bound search has been employed to find the global optimum. In our visual tracking application, we simplify this problem using discrete sampling. That is to say, only a certain number of sampled image patches are evaluated to find the most violated constraint at each column generation iteration. It is also the case for the final inference step for prediction. This simple search strategy has been used in [1].

3.6 StructBoost for CRF parameter learning

CRF has found many applications in computer vision such as image segmentation. However, the parameter learning of CRF remains an issue in many applications. Most work uses tedious cross-validation to find the optimal values for a *small* number of parameters. Recently, structured SVM [14], [16] and a tree-based graph learning method [23] have been proposed to learn these parameters in a principled way. We demonstrate CRF parameter learning using StructBoost in the application of image segmentation. Later we run some experiments on the Graz-02 image segmentation dataset.

To speed up computation, super-pixels rather than pixels have been widely adopted in image segmentation. We define $\boldsymbol{x}$ as an image, $\boldsymbol{y}$ as the segmentation labels of all super-pixels in an image.

We consider the energy E of an image $\boldsymbol{x}$ and segmentation labels $\boldsymbol{y}$ over the nodes $\mathcal{N}$ and edges $\mathcal{S}$, which takes the following form:

$$E(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{w}) = \sum_{p \in \mathcal{N}} \boldsymbol{w}^{(1)} \mathbb{h}^{(1)} \left(\mathbf{U}(y^p, \boldsymbol{x}) \right) + \sum_{(p,q) \in \mathcal{S}} \boldsymbol{w}^{(2)} \mathbb{h}^{(2)} \left(\mathbf{V}(y^p, y^q, \boldsymbol{x}) \right). \tag{21}$$

Here p and q are the super-pixel indexes; and y^p, y^q are the labels of the super-pixels p, q. $\mathbf{U}$ is a set of unary potential functions: $\mathbf{U} = [U_1, U_2, \dots]^\top$. $\mathbf{V}$ is a set of pairwise potential functions: $\mathbf{V} = [V_1, V_2, \dots]^\top$. Details about how to obtain $\mathbf{U}$ and $\mathbf{V}$ are postponed to the experiment section. $\boldsymbol{w}^{(1)}$ and $\boldsymbol{w}^{(2)}$ are the CRF parameters that we want to learn. $\mathbb{h}^{(1)}(\cdot)$ and $\mathbb{h}^{(2)}(\cdot)$ are two sets of weak learners for the unary part and pairwise part respectively: $\mathbb{h}^{(1)}(\cdot) = [\, \hbar_1^{(1)}(\cdot), \hbar_2^{(1)}(\cdot), \dots, \hbar_n^{(1)}(\cdot) \,]^\top$, $\mathbb{h}^{(2)}(\cdot) = [\, \hbar_1^{(2)}(\cdot), \hbar_2^{(2)}(\cdot), \dots, \hbar_n^{(2)}(\cdot) \,]^\top$. In our experiments, we use discrete weak learners and a weak learner $\hbar(\cdot)$ here maps a vector to $\{0, 1\}$, which is different from other experiments. Thus the energy value is always positive: $E \geq 0$. Note that our setting (21) differs most CRF learning settings such as [16]. These traditional CRF methods often use a linear model [16]. Until recently, Bertelli et al. presented an image segmentation approach that uses *nonlinear* kernel for the unary energy term in the CRF model [14]. In our model (21), nonlinearity is introduced by applying weak learners on the potential functions' outputs $\mathbf{U}$ and $\mathbf{V}$. This is in spirit same as the fact that an SVM introduces nonlinearity via the so-called kernel trick and boosting learns a nonlinear model with nonlinear weak learners.

To predict the segmentation labels $\boldsymbol{y}^\star$ of an unknown image $\boldsymbol{x}$ is to solve an energy minimization problem:

$$\boldsymbol{y}^\star = \operatorname*{argmin}_{\boldsymbol{y}} E(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{w}), \tag{22}$$

which can be solved efficiently by graph cuts [16], [24]. To learn the parameters in StructBoost framework, we define $\boldsymbol{w} = -(\boldsymbol{w}^{(1)} \odot \boldsymbol{w}^{(2)})$ and the function $\mathbb{h}(\cdot, \cdot)$:

$$\mathbb{h}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{p \in \mathcal{N}} \mathbb{h}^{(1)} \left(\mathbf{U}(\boldsymbol{y}^p, \boldsymbol{x}) \right) \odot \sum_{(p,q) \in \mathcal{S}} \mathbb{h}^{(2)} \left(\mathbf{V}(\boldsymbol{y}^p, y^q, \boldsymbol{x}) \right). \tag{23}$$

Recall that $\odot$ stacks two vectors. With this definition, we have the relation: $\boldsymbol{w}^\top \mathbb{h}(\boldsymbol{x}, \boldsymbol{y}) = -E(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{w})$. By substituting it into our StructBoost in (3), the CRF parameter learning problem can be written as:

$$\min_{\boldsymbol{w}, \boldsymbol{\xi}} \ \|\boldsymbol{w}\|_1 + \tfrac{C}{m} \sum_i \xi_i \tag{24a}$$

$$\text{s.t.:} \ E(\boldsymbol{x}_i, \boldsymbol{y}; \boldsymbol{w}) - E(\boldsymbol{x}_i, \boldsymbol{y}_i; \boldsymbol{w}) \geq \Delta(\boldsymbol{y}_i, \boldsymbol{y}) - \xi_i, \quad \forall i = 1, \dots, m; \ \text{and} \ \forall \boldsymbol{y} \in \mathcal{Y} \backslash \boldsymbol{y}_i. \ \ \boldsymbol{w} \geq 0; \boldsymbol{\xi} \geq 0. \tag{24b}$$

Here i indexes images. Intuitively, the optimization in (24) is to encourage the energy of the ground truth label $E(\boldsymbol{x}_i, \boldsymbol{y}_i)$ to be lower then any other *incorrect* labels $E(\boldsymbol{x}_i, \boldsymbol{y})$ by at least a margin $\Delta(\boldsymbol{y}_i, \boldsymbol{y})$, $\forall \boldsymbol{y} \neq \boldsymbol{y}_i$. This optimization can be solved in the StructBoost framework using the one-slack algorithm which we discussed before. We use decision stumps for function $\hbar^{(1)}$ and $\hbar^{(2)}$. In each column generation iteration (Algorithm 1), two new weak learners ($\hbar^{(1)\star}$ and $\hbar^{(2)\star}$) are generated and added to unary weak learner set $\mathbb{h}^{(1)}$ and pairwise weak learner set $\mathbb{h}^{(2)}$ respectively by solving the argmax problem defined in (11), which can be written as:

$$\hbar^{(1)\star}(\cdot, \cdot) = \operatorname*{argmax}_{\hbar(\cdot,\cdot)} \sum_{i, \boldsymbol{y} \neq \boldsymbol{y}_i} \sum_{\boldsymbol{c}} \lambda_{(\boldsymbol{c}, \boldsymbol{y})} c_i \cdot \sum_{p \in \mathcal{N}} \Big[\hbar^{(1)} \left(\mathbf{U}(y^p, \boldsymbol{x}_i) \right) - \hbar^{(1)} \left(\mathbf{U}(y_i^p, \boldsymbol{x}_i) \right) \Big]; \tag{25}$$

and,

$$\hbar^{(2)\star}(\cdot, \cdot) = \operatorname*{argmax}_{\hbar(\cdot,\cdot)} \sum_{i, \boldsymbol{y} \neq \boldsymbol{y}_i} \sum_{\boldsymbol{c}} \lambda_{(\boldsymbol{c}, \boldsymbol{y})} c_i \cdot \sum_{(p,q) \in \mathcal{S}} \Big[\hbar^{(2)} \left(\mathbf{V}(y^p, y^q, \boldsymbol{x}_i) \right) - \hbar^{(2)} \left(\mathbf{V}(y_i^p, y_i^q, \boldsymbol{x}_i) \right) \Big]. \tag{26}$$

Considering the maximization to find the most violated constraint corresponding to $\boldsymbol{x}_i$ in line 7 of Algorithm 2:

$$\boldsymbol{y}_i' = \operatorname*{argmax}_{\boldsymbol{y}} \ \Delta(\boldsymbol{y}_i, \boldsymbol{y}) - \boldsymbol{w}^\top \delta \mathbb{h}_i(\boldsymbol{y}). \tag{27}$$

Solving (27) is to solve the inference:

$$\boldsymbol{y}_i' = \underset{\boldsymbol{y}}{\operatorname{argmin}}\ E(\boldsymbol{x}_i, \boldsymbol{y}) - \Delta(\boldsymbol{y}_i, \boldsymbol{y}), \tag{28}$$

which is similar to the label prediction inference in (22), and the only difference is that the labeling loss term: $\Delta(\boldsymbol{y}_i, \boldsymbol{y})$ is involved in (28). We simply define $\Delta(\boldsymbol{y}_i, \boldsymbol{y})$ using Hamming loss, which is the sum of the differences between the ground truth label $\boldsymbol{y}_i$ and the label $\boldsymbol{y}$ over super-pixels:

$$\Delta(\boldsymbol{y}_i, \boldsymbol{y}) = \sum_p (1 - \mathbb{I}(y_i^p, y^p)). \tag{29}$$

$\mathbb{I}(\cdot, \cdot)$ is an indicator function defined in (15). With this definition, the term $\Delta(\boldsymbol{y}_i, \boldsymbol{y})$ can be absorbed into the unary term of the energy function defined in (21). The inference in (28) can be written as:

$$\begin{aligned}\boldsymbol{y}_i' = \underset{\boldsymbol{y}}{\operatorname{argmin}} & \sum_{p \in \mathcal{N}} \left[\boldsymbol{w}^{(1)} \mathbb{h}^{(1)} \left(\mathbf{U}(\cdot) \right) - (1 - \mathbb{I}(y_i^p, y^p)) \right] \\ & + \sum_{(p,q) \in \mathcal{S}} \boldsymbol{w}^{(2)} \mathbb{h}^{(2)} \left(\mathbf{V}(\cdot) \right). \end{aligned} \tag{30}$$

Similar to [16], the minimization (30) still can be solved efficiently by graph cuts.

4 Experiments

In this section, we run various experiments on applications including binary classification, ordinal regression, multi-class image classification, hierarchical image classification, visual tracking and image segmentation. We use UCI datasets in binary classification and ordinal regression for training time comparison, we have randomly chosen 75% data as training data and the rest 25% for test. For each dataset we run 10 times and report the average results. We use 4-fold cross validation on the entire dataset to determine the regularization parameter. The value of the regularization parameter C is chosen from 2^2 to 2^6. For all the experiments, we have set the cutting-plane ϵ_{cp} to 0.01. The threshold of the column generation stopping criterion ϵ_{cg} is 0.01. Maximum column generation iteration is set to 200. The CPU time is obtained on a desktop with an AMD CPU 2.20GHz. The code is in Matlab that calls some C mex files.

4.1 Binary classification

We run experiments on some UCI machine learning datasets to compare our StructBoost formulation of binary boosting against the standard LPBoost [9]. Table 1 reports the experiment results on binary classification data sets (we use one class as positive data and the rest classes as negative if the original datasets have multiple labels). It is easy to see that the 1-slack formulation is orders of magnitude faster than the standard LPBoost.

dataset	method	CPU time (s)	training %	test %
svmguide4	LPBoost	23±5	2.0±0.7	5.3±1.8
	1-slack	2.6±0.7	1.9±0.4	5.1±1.3
vehicle	LPBoost	196±31	13.7±1.0	21.8±2.0
	1-slack	55±10	14.1±1.5	21.0±2.7
dna	LPBoost	1818±476	2.6±0.4	4.6±1.0
	1-slack	92±13	2.7±0.3	4.4±0.9
segment	LPBoost	1345±282	0.7±0.1	0.8±0.3
	1-slack	0.4±0.1	0.6±0.3	0.8±0.3
satimage	LPBoost	$>$ 8h	1.7±0.1	1.9±0.4
	1-slack	121±11	1.8±0.1	1.9±0.3
waveform	LPBoost	$>$ 8h	8.5±0.2	10.6±0.9
	1-slack	106±9	8.5±0.3	10.8±0.8
banana	LPBoost	11783±2786	27.8±0.2	28.5±0.8
	1-slack	0.8±0.1	27.8±0.3	28.4±0.8
twonorm	LPBoost	$>$ 8h	5.1±0.8	6.0±0.7
	1-slack	444±46	2.9±0.2	3.9±0.5
usps	LPBoost	$>$ 8h	2.7±0.2	3.0±0.5
	1-slack	88±21	1.0±0.1	1.3±0.2
pendigits	LPBoost	−	−	−
	1-slack	7±1	3.8±0.1	3.8±0.1

TABLE 1: Binary classification. We compare the 1-slack StructBoost formulation of binary boosting agaisnt standard LPBoost [9] (i.e., m-slack formulation). We report the training CPU time (in seconds), training and test error (in percentage). The speedup is significant, especially on large-scale datasets. "−" means no results obtained or the number of completed column generation iterations being less than 5 after running 8 hours. "$>$" means that the method is not converged after running 8 hours.

dataset	method	time (s)	AUC training	AUC test
wine	m-slack	2850±480	1.000±0.000	0.992±0.007
	1-slack	0.2±0.1	0.998±0.001	0.991±0.007
glass	m-slack	$>$ 8h	0.994±0.004	0.902±0.047
	1-slack	29±10	1.000±0.000	0.876±0.028
svmguide2	m-slack	−	−	−
	1-slack	72±17	0.987±0.005	0.895±0.027
svmguide4	m-slack	−	−	−
	1-slack	11±4	0.998±0.001	0.981±0.011
vehicle	m-slack	−	−	−
	1-slack	1426±369	0.938±0.006	0.840±0.019
dna	m-slack	−	−	−
	1-slack	31±6	0.988±0.002	0.987±0.005
segment	m-slack	−	−	−
	1-slack	27±4	1.000±0.000	0.996±0.002
satimage	m-slack	−	−	−
	1-slack	11938±2100	0.999±0.000	0.986±0.002

TABLE 2: AUC maximization. We compare the performance of m-slack and 1-slack formulations. "−" means no results obtained or the number of completed column generation iterations being less than 5 after running 8 hours. "$>$" means that the method is not converged after running 8 hours. It clearly shows that 1-slack is significantly faster.

4.2 Ordinal regression and AUC optimization

The details of StructBoost for AUC optimization are described in Section 3.4. We run AUC optimization with the m-slack formulation of StructBoost and (solving (3) or its dual) 1-slack formulation of StructBoost (solving (7)). To create imbalanced data, we have used one class of the multi-class UCI datasets as positive data and all the rest labels as negative data. Table 2 reports the results. We can see that the 1-slack formulation of StructBoost is much faster with similar performance.

Note that RankBoost may also be applied to this problem [25]. RankBoost has been designed for solving ranking problems and it is not a general structured boosting method.

4.3 Multi-class classification

The details of StructBoost for multi-class are described in Section 3.2. We run our multi-class boosting on two image datasets: MNIST[2] and Scene15 [26]. Here we have used the linear ℓ_1 SVM as weak classifiers. We set the trade-off parameter as $C = 10^6/(\text{number of examples})$. To avoid over-fitting, at each boosting iteration, we first sort the data weights and select top p percentage of the weighted positive and negative examples to train the SVM ($p = 60\%$ for MNIST and 80% for Scene15).

For MNIST, we randomly select 100 samples from each class as training sets and use the original test sets of $10,000$ samples. We have repeated this procedure for 5 times and reported the average test error. Spatial pyramid HOG features [27] are used here. For Scene15, we randomly sample 100 examples of each class to generate training data, and the rest as testing data. So the total number of training examples is 1500, and the number of test examples is 2985. The reported result is the average of 5 runs. We generate histograms of code words as features. The code book size is 200. An image is divided into 31 sub-windows in a spatial hierarchy manner [28]. We generate histograms in each sub-windows, so the histogram feature dimension is 6200. CENTRIST [29] is used as the feature descriptor. In each train/test split, a visual codebook is generated using only training images. Both training and test images are then transformed into histograms of code words.

For comparison, we also run two standard multi-class boosting methods: AdaBoost.ECC [30] and AdaBoost.MH [31]. We use decision stumps for AdaBoost.MH and AdaBoost.ECC. Figure 2 shows the convergence curves. The observations are: 1) linear SVM as weak classifiers seems to converge faster than decision stumps. 2) Our StructBoost converges faster than other competitors, although the final accuracy is not significantly different from others.

4.4 Hierarchical multi-class classification

The details of hierarchical multi-class are described in Section 3.3. We have constructed two hierarchical image datasets from the SUN dataset [22]. The first dataset contains 6 classes of scenes, it has two category levels. For each scene class, we use the top first 200 images from the original SUN dataset. So there are 1200 images in total. The second dataset is larger which contains 15 classes of scenes, and there are 3000 images in total. We have used the HOG features as described in [22]. The detail of the hierarchical structure of these two dataset is show in the Figure 1.

For each dataset, we randomly select 50% examples for training, and the rest for testing. The reported results are computed on 8 random splits. We heuristically set the regularization parameter for the StructBoost in this experiment. The maximum boosting iteration is set to 500.

Table 3 reports the results. Here we have also run standard multi-class boosting, AdaBoost.ECC, and AdaBoost.MH. Two observations can be made: 1) Hierarchical multi-class boosting indeed has the minimum tree loss over all the compared methods because it directly minimizes the tree loss; 2) Hierarchical multi-class boosting improves its standard multiclass counterpart (the second column in Table 3) in terms of both classification accuracy and the tree loss, demonstrating its usefulness.

2. http://yann.lecun.com/exdb/mnist/

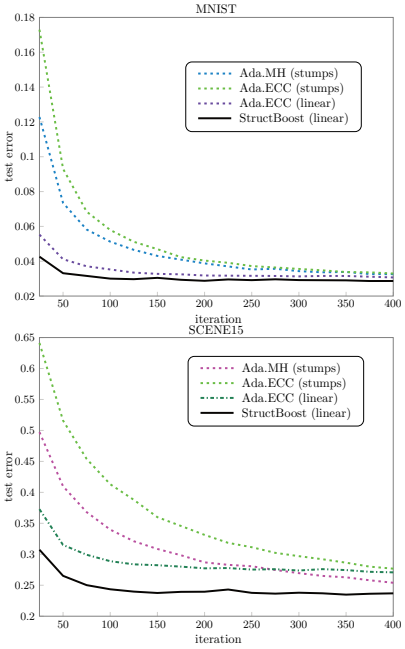


Fig. 2: We compare StructBoost with AdaBoost.ECC [30] and AdaBoost.MH [31] on two image multi-class classification datasets: MNIST and Scene15. "Stumps" means decision stumps as weak learners and "linear" means linear ℓ_1 SVM as weak learners. Our method performs the best in terms of convergence rate and accuracy.

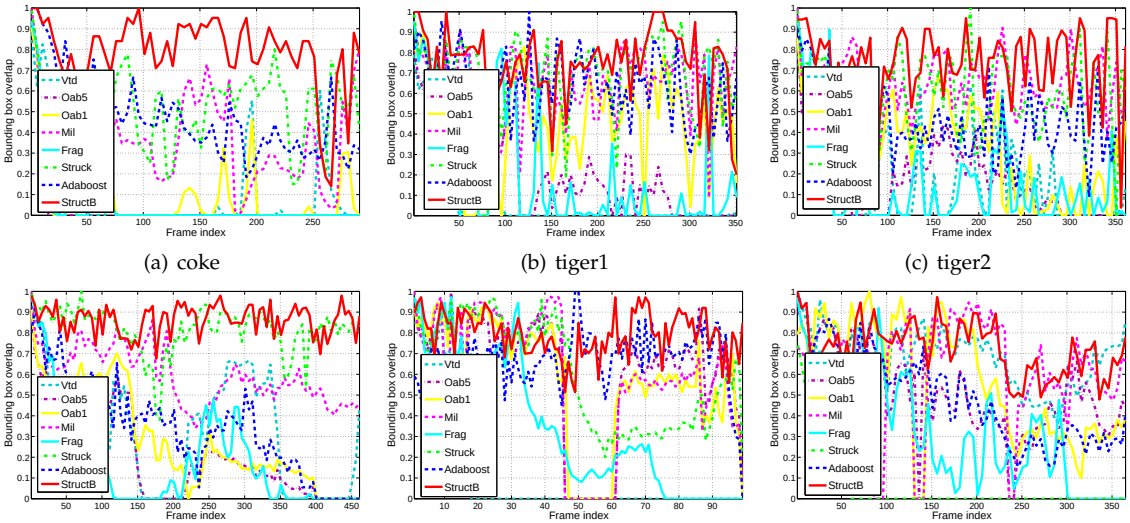
4.5 Visual tracking by optimizing the image area overlap criterion

In [1], a visual tracking method, termed Struck, was introduced based on SSVM. The core idea is to train a tracker by optimizing the Pascal image overlap score using SSVM. Here we follow the same general setting of this structured tracking method, but with our StructBoost, instead of SSVM. We use decision stumps as the weak learner. More details are described in Section 3.5.

We use an on-line tracking setting for StructBoost tracker in our experiment. We only use the first 3 labeled frames for initialization and training our StructBoost tracker. We then update our tracker by re-training the model with sequent frames during the course of tracking. In the i-th frame (represented by $\boldsymbol{x}_i$), we first perform a prediction step to output the detection box, then collect training data for tracker update. In the prediction step, we solve the inference in (2) to output the prediction box (represented by $\boldsymbol{y}_i$) of current frame. For solving the inference in (2), we simply sample about 2000 bounding boxes around the prediction bounding box of last frame (represented by $\boldsymbol{y}_{i-1}$), one sampled bounding box is denoted by $\boldsymbol{y}$, and search the most confident bounding box over all sampled boxes $\boldsymbol{y}$ as the prediction $\boldsymbol{y}_i$. In the first 3 labelled frames for initialization, we use the labelled bounding box as $\boldsymbol{y}_i$.

	StructBoost (tree loss)	StructBoost (flat)	Ada.ECC (flat)	Ada.ECC (flat, stumps)	Ada.MH (flat, stumps)
6 scenes (error rate %)	32.9±1.2	35.1±1.0	34.4±1.5	32.9±1.5	**32.2±0.8**
6 scenes (tree loss)	**0.345±0.015**	0.389±0.013	0.368±0.016	0.352±0.015	0.351±0.011
15 scenes (error rate %)	44.7±1.1	45.3±0.8	**43.6±0.8**	45.0±1.2	43.7±0.8
15 scenes (tree loss)	**0.519±0.011**	0.551±0.015	0.533±0.009	0.529±0.015	0.527±0.015

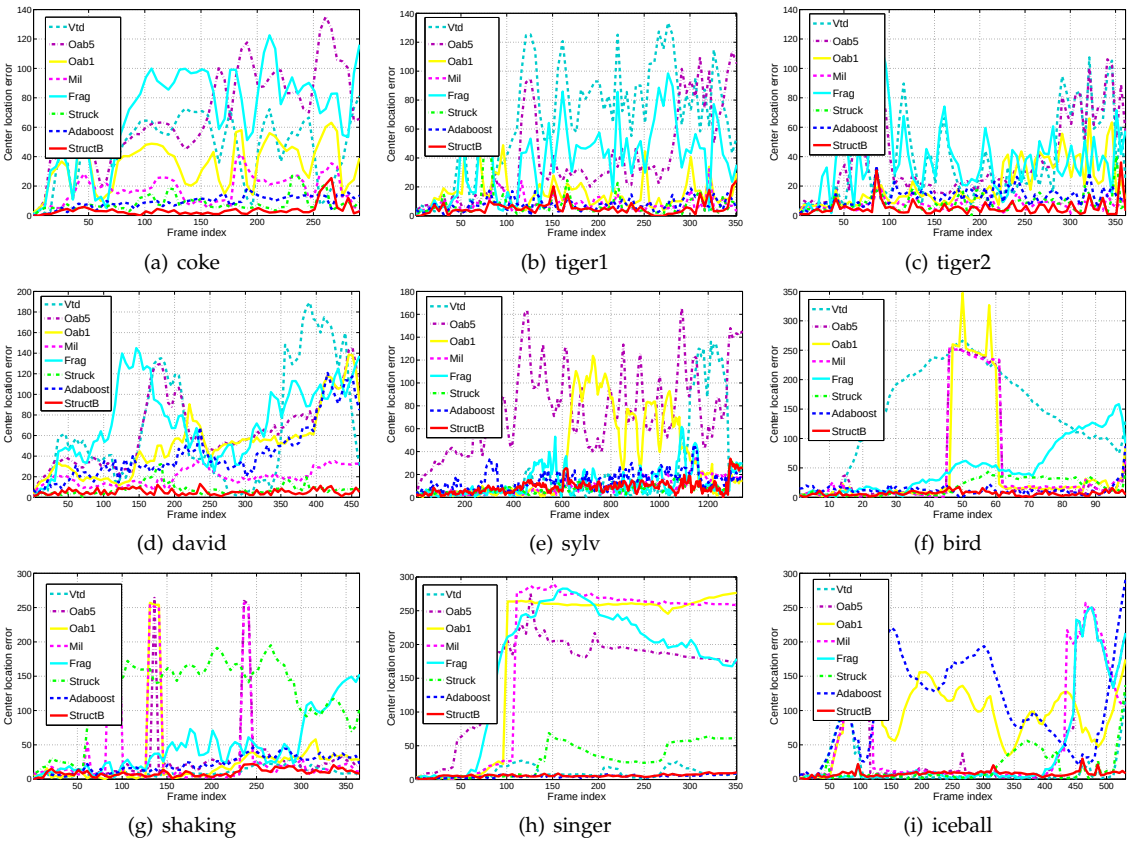
TABLE 3: Hierarchical classification. Results on subsets of the SUN dataset. The first three boosting classifiers use linear SVM as weak classifiers and the latter two use decision stumps. The first one is the structured optimization of the tree loss and all the others optimize conventional multi-class classification losses ('flat' loss). StructBoost that directly minimizes the tree loss indeed performs best in the tree loss.



(a) coke (b) tiger1 (c) tiger2

(d) david (e) bird (f) shaking

Fig. 3: Bounding box overlap in frames of several video sequences. We compare our StructBoost tracker with a few state-of-the-art trackers as well as the binary AdaBoost tracker. Results show that in most case StructBoost has higher overlap scores hence performs the best.



(a) coke (b) tiger1 (c) tiger2

(d) david (e) sylv (f) bird

(g) shaking (h) singer (i) iceball

Fig. 4: Center location error (pixels) in frames of several video sequences. We compare our StructBoost tracker with a few state-of-the-art trackers as well as the binary AdaBoost tracker. Results show that in most case StructBoost has lower center location error hence performs the best.

Fig. 5: Some tracking examples of several video sequences: "coke", "david", "bird" and "walk". The output bounding boxes of our StructBoost usually have better overlap with the target object then other methods.

	StructBoost	AdaBoost	Struck_{50}	Frag	MIL	OAB_1	OAB_5	VTD
coke	**0.79 ± 0.17**	0.47 ± 0.19	0.55 ± 0.18	0.07±0.21	0.36±0.23	0.10 ± 0.20	0.04 ± 0.16	0.10 ± 0.23
tiger1	**0.75 ± 0.17**	0.64 ± 0.16	0.68 ± 0.21	0.21±0.30	0.64±0.18	0.44 ± 0.23	0.23 ± 0.24	0.11 ± 0.24
tiger2	**0.74 ± 0.18**	0.46 ± 0.18	0.59 ± 0.19	0.16±0.24	0.63±0.14	0.35 ± 0.23	0.18 ± 0.19	0.19 ± 0.22
david	**0.86 ± 0.07**	0.34 ± 0.23	0.82 ± 0.11	0.18±0.24	0.59±0.13	0.28±0.23	0.21±0.22	0.29 ± 0.27
girl	0.74 ± 0.12	0.41 ± 0.26	**0.80 ± 0.10**	0.65±0.19	0.56±0.21	0.43±0.18	0.28±0.26	0.63 ± 0.12
sylv	0.66 ± 0.16	0.52 ± 0.18	**0.69 ± 0.14**	0.61±0.23	0.66±0.18	0.47 ± 0.38	0.05 ± 0.12	0.58 ± 0.30
bird	**0.79 ± 0.11**	0.67 ± 0.14	0.60 ± 0.26	0.34±0.32	0.58±0.32	0.57 ± 0.29	0.59 ± 0.30	0.11 ± 0.26
walk	**0.74 ± 0.19**	0.56 ± 0.14	0.59 ± 0.39	0.09±0.25	0.51±0.34	0.54 ± 0.36	0.49 ± 0.34	0.08 ± 0.23
shaking	**0.72 ± 0.13**	0.49 ± 0.22	0.08 ± 0.19	0.33±0.28	0.61±0.26	0.57 ± 0.28	0.51 ± 0.21	0.69 ± 0.14
singer	0.69 ± 0.10	**0.74 ± 0.10**	0.34 ± 0.37	0.14±0.30	0.20±0.34	0.20±0.33	0.07 ± 0.18	0.50 ± 0.20
iceball	**0.58 ± 0.17**	0.05 ± 0.16	0.51 ± 0.33	0.51±0.31	0.35±0.29	0.08±0.23	0.38 ± 0.30	0.57 ± 0.29

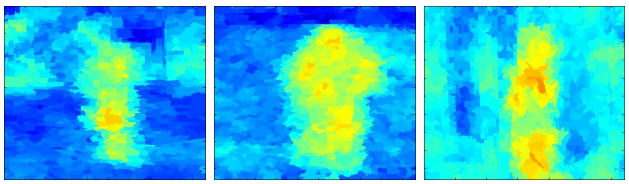
TABLE 4: Average bounding box overlap scores on benchmark videos. Both StructBoost and AdaBoost use decision stumps trained on raw pixels and HOG features. Struck_{50} is structured SVM tracking with a buffer size of 50 [1]. Our StructBoost outperforms other methods on all the sequences. Structured SVM of [1] is the second best, which confirms the usefulness of structured training.

	StructBoost	AdaBoost	Struck_{50}	Frag	MIL	OAB_1	OAB_5	VTD
coke	**3.7 ± 4.5**	9.3 ± 4.2	8.3 ± 5.6	69.5±32.0	17.8±9.6	34.7 ± 15.5	68.1 ± 30.3	46.8 ± 21.8
tiger1	**5.4 ± 4.9**	7.8 ± 4.4	7.8 ± 9.9	39.6±25.7	8.4±5.9	17.8 ± 16.4	38.9 ± 31.1	68.8 ± 36.4
tiger2	**5.2 ± 5.6**	12.7 ± 6.3	8.7 ± 6.1	38.5±24.9	7.5±3.6	20.5 ± 14.9	38.3 ± 26.9	38.0 ± 29.6
david	**5.2 ± 2.8**	43.0 ± 28.2	7.7 ± 5.7	73.8±36.7	19.6±8.2	51.0±30.9	64.4±33.5	66.1 ± 56.3
girl	14.3 ± 7.8	47.1 ± 29.5	**10.1 ± 5.5**	23.0±22.5	31.6±28.2	43.3±17.8	67.8±32.5	18.4 ± 11.4
sylv	9.1 ± 5.8	14.7 ± 7.8	**8.4 ± 5.3**	12.2±11.8	9.4±6.5	32.9 ± 36.5	76.4 ± 35.4	21.6 ± 35.7
bird	**6.7 ± 3.8**	12.7 ± 9.5	17.9 ± 13.9	50.0±43.3	49.0±85.3	47.9 ± 87.7	48.5 ± 86.3	143.9 ± 79.3
walk	**8.4 ± 10.3**	13.5 ± 5.4	33.9 ± 49.5	102.8±46.3	35.0±47.5	35.7 ± 49.2	38.0 ± 48.7	100.9 ± 47.1
shaking	**9.5 ± 5.4**	21.6 ± 12.0	123.9 ± 54.5	47.2±40.6	37.8±75.6	26.9 ± 49.3	29.1 ± 48.7	10.5 ± 6.8
singer	5.8 ± 2.2	**4.8 ± 2.1**	29.5 ± 23.8	172.8±95.2	188.3±120.8	189.9 ± 115.2	158.5 ± 68.6	10.1 ± 7.6
iceball	**8.0 ± 4.1**	107.9 ± 66.4	15.6 ± 22.1	39.8±72.9	61.6±85.6	97.7 ± 53.5	58.7 ± 84.0	13.5 ± 26.0

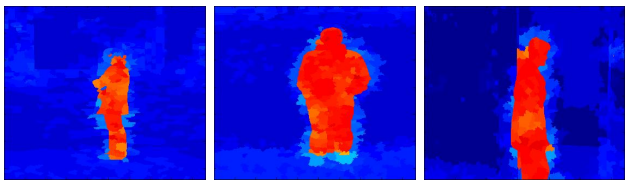
TABLE 5: Average center errors on benchmark videos. Both StructBoost and AdaBoost use decision stumps trained on raw pixels and HOG features. Struck_{50} is structured SVM tracking with a buffer size of 50 [1]. We observe similar results as in Table 4: Our StructBoost outperforms other methods on all the sequences, and structured SVM of [1] is the second best. This again confirms the usefulness of structured training.

(a) Testing images

(b) Ground truth



(c) AdaBoost confidence map

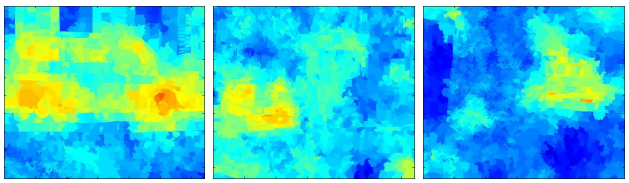


(d) StructBoost confidence map, 2 unary and 2 pairwise potentials

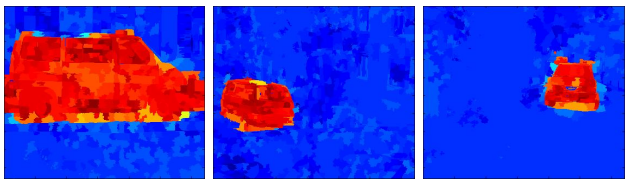
Fig. 6: Person image segmentation examples on Graz-02 dataset. Confidence scores are normalized to $[-1, 1]$ for all methods. Red color indicates strong confidence for foreground while blue indicates strong confidence for background. Compared to the AdaBoost which only use 1 unary potential, our StructBoost, which combines unary and smooth pairwise potentials by parameter learning, has sharper boundary, better spatial regularization, and higher confidences of target objects.

(a) Testing images

(b) Ground truth



(c) AdaBoost confidence map



(d) StructBoost confidence map, 2 unary and 2 pairwise potentials

Fig. 7: Car image segmentation examples on Graz-02 dataset. Our StructBoost, which combines unary and smooth pairwise potentials by parameter learning, has sharper boundary, better spatial regularization, and higher confidences of target objects.. See Figure 6 for more details.

After the prediction, we collect training data by sampling about 200 bounding boxes around the current prediction $\boldsymbol{y}_i$. We use the training data in recent 60 frames to re-train the tracker for every 2 frames. We search over those sampled bounding boxes for finding the most violated constraint of each frame in the training process, which analogue to the prediction inference.

For StructBoost, the maximum number of weak learners is set to 300. The regularization parameter is selected from $10^{0.5}$ to 10^2. We use the down-scaled gray-scale raw pixels and HOG as image features. For HOG feature, we use the code in [32]. For comparison, we also run the AdaBoost trackers using the same setting as our StructBoost tracker. For AdaBoost training, the maximum number of weak learners is set to 500. The AdaBoost tracker is a simple binary model. When updating (or initializing) AdaBoost tracker, we collect positive training boxes that significantly overlap with the predicted bounding box of the current frame (overlap above 0.8), and negative training boxes with small overlap (overlap lower or equal to 0.3).

We also compare our trackers with a few state-of-the-art tracking methods, including Struck [1] (with a buffer size of 50), multi-instance tracking (MIL) [33], fragment tracking (Frag) [34], online AdaBoost tracking (OAB) [35], and visual tracking decomposition (VTD) [36]. OAB has two versions with two different settings ($r = 1$ means only one positive example per frame and $r = 5$ means five positive examples per frame for training. They are referred to as OAB_1 and OAB_5 here. See [33].). The test video sequences "coke, tiger1, tiger2, david, girl and sylv" were used in [1]. The sequences "shaking, singer" are obtained from [36], and the rest sequences are from [37].

Table 4 reports the Pascal overlap score of various tracking methods on testing video sequences. *Our StructBoost tracker performs best on most test sequences*. Compared with the binary AdaBoost tracker, StructBoost has a significantly higher score. Note that here Struck uses Haar features. When Struck uses a Gaussian kernel defined on raw pixels, the performance is slightly different [1], and ours still outperforms Struck in most cases. This might be due to the fact that our StructBoost selects relevant features (300 features selected here), and the SSVM of [1] uses all the image patch information which may contain noises.

Figure 3 plots the Pascal overlap scores frame by frame on several video sequences. It clearly shows that StructBoost outperform other methods in most cases. Compared to AdaBoost, StructBoost performs better at almost all

Evaluation	precision=recall (%)			intersection/union (%)		
Category	bike	car	people	bike	car	people
SVM	68.0	63.4	61.1	65.0	68.9	62.8
AdaBoost	72.7	67.8	67.0	69.2	72.2	68.9
StructBoost-CRF	**74.9**	**72.4**	**72.6**	**71.8**	**76.0**	**72.7**

TABLE 6: Image segmentation results on the Graz-02 dataset. The precision=recall point [24] and intersection-union score are used to evaluation our method. The result shows that StructBoost with the efficient graph-cuts inference is able to learn the CRF parameters in a principled way, and improves the performance.

frames. The main reason is that StructBoost directly maximizes the overlap, while AdaBoost is trained by optimizing the classification error, which is not directly related to the Pascal overlap score.

The central location errors of compared methods are shown in Table 5. Our method also achieve the best results in most cases, which reveals that optimizing the overlap score also helps minimize the central location errors. We also plot the central location errors of different methods frame by frame on several sequences in Figure 4. These results prove the superior performance of StructBoost for tracking.

Some tracking examples are shown in Figure 5. In our experiments, the output space of StructBoost is the bounding box's coordinates and the scale is fixed. However, it is easy to incorporate scale changes, rotation and transforms into the output space due to the flexibility of StructBoost.

4.6 CRF parameter learning for image segmentation

In this experiment, we extend the super-pixels based segmentation method [24] with CRF parameter learning. More details are described in Section 3.6. We use the Graz-02 dataset[3] in this experiment, which contains 3 categories (bike, car and person). Each image only contains one category. For each category, we use first 300 labeled images. Images with the odd indices are for training and the rest for testing. We generate super-pixels and features same as in [24]: the neighborhood size is set to 2; histogram of visual words features are generated for each super-pixel; code book size is 200. For StructBoost, we use two unary potentials: $\mathbf{U} = [\, U_1, U_2 \,]^\top$ and 2 pairwise potentials: $\mathbf{V} = [\, V_1, V_2 \,]^\top$. We only use randomly sampled 50 training images for the training of StructBoost to learn CRF parameters. In binary classifier training for the unary potential, we use all training images.

Two unary potentials: U_1, U_2 are constructed using two AdaBoost classifiers; one is trained on the visual word histogram features [24], and the other is trained on color histogram together with the thumbnail feature [38]. We define F' as the discriminant function of AdaBoost. Then the unary potential function can be written as:

$$U(\boldsymbol{x}, y^p) = -y^p F'(\boldsymbol{x}). \tag{31}$$

For the two pairwise potentials, V_1 is constructed using color difference, and V_2 is constructed using shared boundary length between two neighboring super-pixels [24], which is able to discourage small isolated segments. Recall that $\mathbb{I}(\cdot,\cdot)$ is an indicator function defined in (15). $\|\boldsymbol{x}^p - \boldsymbol{x}^q\|_2$ calculates the ℓ_2 norm of the color difference between two super-pixels in the LUV color-space; $\ell(\boldsymbol{x}^p, \boldsymbol{x}^q)$ is the shared boundary length between two super-pixels, as in [24]. Then V_1, V_2 can be written as:

$$V_1(\cdot) = \exp(-\|\boldsymbol{x}^p - \boldsymbol{x}^q\|_2)\big[1 - \mathbb{I}(y^p, y^q)\big], \tag{32}$$

$$V_2(\cdot) = \ell(\boldsymbol{x}^p, \boldsymbol{x}^q)\big[1 - \mathbb{I}(y^p, y^q)\big]. \tag{33}$$

For comparison, we also run AdaBoost and SVM for segmentation, which are binary classifiers trained on foreground and background super-pixels using the same visual word histogram features as our method. As [24], we use the $precision = recall$ point and intersection-union score to evaluation our method. Results are shown in Table 6. Some segmentation examples are shown in Figures 6 and 7. The results show that StructBoost with the efficient inference method (graph cuts) gains performance improvement, and also show that StructBoost is able to learn the CRF parameters for combining different potential functions in a principled way.

5 Conclusion

We have presented a boosting method for structural learning, as an alternative to SSVM [4] and CRF [10]. Analogues to SSVM, where the discriminant function is learned over a joint feature space of inputs and outputs, the discriminant function of the proposed StructBoost is a linear combination of weak learners defined over a joint space of input-output pairs.

Moreover, StructBoost is flexible in its ability to optimize specific loss functions. To efficiently solve the resulting optimization problems, we have introduced a cutting-plane method, which was originally proposed for fast training of linear SVM. Our extensive experiments demonstrate that indeed the proposed algorithm is computationally tractable. We also show that the test accuracy of our StructBoost is at least comparable or sometimes exceeds conventional approaches for a wide range of applications such as multi-class classification, AUC optimization, image segmentation with CRF parameter learning. In particular, we have used StructBoost to train a visual tracker by optimizing the Pascal image overlap score. Experiments show its state-of-the-art tracking accuracy, compared with a few recent tracking methods. Future work will focus on more applications of this general StructBoost framework.

References

[1] S. Hare, A. Saffari, and P. Torr, "Struck: Structured output tracking with kernels," in *Proc. IEEE Int. Conf. Comp. Vis.*, 2011.

[2] S. Nowozin and C. H. Lampert, "Structured learning and prediction in computer vision," *Foundations & Trends in Computer Graphics & Vision*, 2011.

[3] M. B. Blaschko and C. H. Lampert, "Learning to localize objects with structured output regression," in *Proc. Eur. Conf. Comp. Vis.*, 2008, pp. 2–15.

[4] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun, "Support vector machine learning for interdependent and structured output spaces," in *Proc. Int. Conf. Mach. Learn.*, 2004, pp. 104–111.

3. http://www.emt.tugraz.at/~pinz/

[5] J. Weston and C. Watkins, "Multi-class support vector machines," in *Proc. Euro. Symp. Artificial Neural Networks*, 1999.

[6] K. Crammer and Y. Singer, "On the algorithmic implementation of multiclass kernel-based vector mchines," *J. Mach. Learn. Res.*, vol. 2, pp. 265–292, 2001.

[7] C. Shen and H. Li, "On the dual formulation of boosting algorithms," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 12, pp. 2216–2231, 2010.

[8] T. Joachims, "Training linear SVMs in linear time," in *Proc. ACM SIGKDD Int. Conf. Knowledge discovery & data mining*, 2006, pp. 217–226.

[9] A. Demiriz, K. P. Bennett, and J. Shawe-Taylor, "Linear programming boosting via column generation," *Mach. Learn.*, vol. 46, no. 1-3, pp. 225–254, 2002.

[10] J. Lafferty, A. McCallum, and F. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *Proc. Int. Conf. Mach. Learn.*, 2001, pp. 282–289.

[11] C. Shen and Z. Hao, "A direct formulation for totally-corrective multi-class boosting," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2011.

[12] C. Sutton and A. McCallum, "An introduction to conditional random fields," *Foundations and Trends in Machine Learning*, 2012. [Online]. Available: http://arxiv.org/abs/1011.4088

[13] N. Plath, M. Toussaint, and S. Nakajima, "Multi-class image segmentation using conditional random fields and global classification," in *Proc. Int. Conf. Mach. Learn.*, 2009.

[14] L. Bertelli, T. Yu, D. Vu, and B. Gokturk, "Kernelized structural SVM learning for supervised object segmentation," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.* IEEE, 2011, pp. 2153–2160.

[15] C. Desai, D. Ramanan, and C. C. Fowlkes, "Discriminative models for multi-class object layout," *Int. J. Comp. Vis.*, vol. 95, no. 1, pp. 1–12, 2011.

[16] M. Szummer, P. Kohli, and D. Hoiem, "Learning CRFs using graph cuts," in *Proc. Eur. Conf. Comp. Vis.*, 2008, pp. 582–595.

[17] N. Ratliff, D. Bradley, J. A. Bagnell, and J. Chestnutt, "Boosting structured prediction for imitation learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2007.

[18] L. Mason, J. Baxter, P. L. Bartlett, and M. R. Frean, "Boosting algorithms as gradient descent," in *Proc. Adv. Neural Inf. Process. Syst.*, 1999, pp. 512–518.

[19] C. Parker, "Structured gradient boosting," 2007, PhD thesis, Oregon State University. [Online]. Available: http://hdl.handle.net/1957/6490

[20] Q. Wang, D. Lin, and D. Schuurmans, "Simple training of dependency parsers via structured boosting," in *Proc. Int. Joint Conf. Artificial Intell.*, 2007, pp. 1756–1762.

[21] V. Franc and S. Sonnenburg, "Optimized cutting plane algorithm for support vector machines," in *Proc. Int. Conf. Mach. Learn.*, New York, NY, USA, 2008, pp. 320–327. [Online]. Available: http://doi.acm.org/10.1145/1390156.1390197

[22] J. Xiao, J. Hays, K. Ehinger, A. Oliva, and A. Torralba, "SUN database: Large-scale scene recognition from abbey to zoo," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2010.

[23] S. Nowozin, P. V. Gehler, and C. H. Lampert, "On parameter learning in CRF-based approaches to object class image segmentation," in *Proc. Eur. Conf. Comp. Vis.*, 2010, pp. 98–111.

[24] B. Fulkerson, A. Vedaldi, and S. Soatto, "Class segmentation and object localization with superpixel neighborhoods," in *Proc. Int. Conf. Comp. Vis.*, 2009.

[25] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer, "An efficient boosting algorithm for combining preferences," *J. Mach. Learn. Res.*, vol. 4, pp. 933–969, 2003.

[26] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, vol. 2, 2006, pp. 2169 – 2178.

[27] S. Maji and J. Malik, "Fast and accurate digit classification," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-159, Nov 2009. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-159.html

[28] A. Bosch, A. Zisserman, and X. Munoz, "Scene classification using a hybrid generative/discriminative approach," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 4, pp. 712 – 727, 2008.

[29] J. Wu and J. M. Rehg, "CENTRIST: A visual descriptor for scene categorization," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 8, pp. 1489–1501, 2011.

[30] V. Guruswami and A. Sahai, "Multiclass learning, boosting, and error-correcting codes," in *Proc. Annual Conf. Computational Learning Theory*. ACM, 1999, pp. 145–155.

[31] R. E. Schapire and Y. Singer, "Improved boosting algorithms using confidence-rated predictions," in *Mach. Learn.*, 1999, pp. 80–91.

[32] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 9, pp. 1627–1645, September 2010.

[33] B. Babenko, M.-H. Yang, and S. Belongie, "Visual tracking with online multiple instance learning," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2009.

[34] A. Adam, E. Rivlin, and I. Shimshoni, "Robust fragments-based tracking using the integral histogram," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2006, pp. 798–805.

[35] H. Grabner, M. Grabner, and H. Bischof, "Real-time tracking via on-line boosting," in *Proc. British Mach. Vis. Conf.*, 2006, pp. 47–56.

[36] J. Kwon and K. M. Lee, "Visual tracking decomposition," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2010, pp. 1269–1276.

[37] S. Wang, H. Lu, F. Yang, and M.-H. Yang, "Superpixel tracking," *Proc. Int. Conf. Comp. Vis.*, pp. 1323–1330, 2011.

[38] J. Tighe and S. Lazebnik, "Superparsing: Scalable nonparametric image parsing with superpixels," in *Proc. Eur. Conf. Comp. Vis.*, 2010, pp. 352–365.